Study guide: Finite difference schemes for diffusion processes

Hans Petter Langtangen[1,2]

[1]Center for Biomedical Computing, Simula Research Laboratory
[2]Department of Informatics, University of Oslo

Aug 28, 2014

Contents

1 The 1D diffusion equation

The famous *diffusion equation*, also known as the *heat equation*, reads

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

Here,

- $u(x, t)$: unknown
- α: *diffusion coefficient*

Alternative, compact notation:

$$u_t = \alpha u_{xx}$$

1.1 The initial-boundary value problem for 1D diffusion

$$\frac{\partial u}{\partial t} = \alpha\frac{\partial^2 u}{\partial x^2}, \quad x\in(0,L),\ t\in(0,T] \tag{1}$$
$$u(x,0) = I(x), \qquad x\in[0,L] \tag{2}$$
$$u(0,t) = 0, \qquad t>0, \tag{3}$$
$$u(L,t) = 0, \qquad t>0\,. \tag{4}$$

Note:

- First-order derivative in time: one initial condition
- Second-order derivative in space: a boundary condition at each point of the boundary (2 points in 1D)
- Numerous applications throughout physics and biology

1.2 Step 1: Discretizing the domain

Mesh in time:

$$0 = t_0 < t_1 < t_2 < \cdots < t_{N_t-1} < t_{N_t} = T \tag{5}$$

Mesh in space:

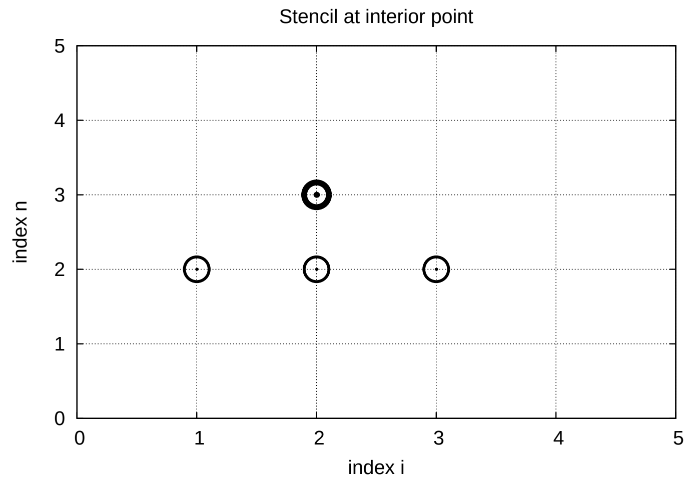
$$0 = x_0 < x_1 < x_2 < \cdots < x_{N_x-1} < x_{N_x} = L \tag{6}$$

Uniform mesh with constant mesh spacings Δt and Δx:

$$x_i = i\Delta x,\ i=0,\ldots,N_x, \quad t_i = n\Delta t,\ n=0,\ldots,N_t \tag{7}$$

1.3 The discrete solution

- The numerical solution is a mesh function: $u_i^n \approx u_{\rm e}(x_i,t_n)$
- Finite difference stencil (or scheme): equation for u_i^n involving neighboring space-time points



1.4 Step 2: Fulfilling the equation at the mesh points

Require the PDE (1) to be fulfilled at an arbitrary *interior mesh point* (x_i,t_n) leads to

$$\frac{\partial}{\partial t}u(x_i,t_n) = \alpha\frac{\partial^2}{\partial x^2}u(x_i,t_n) \tag{8}$$

Applies to all interior mesh points: $i=1,\ldots,N_x-1$ and $n=1,\ldots,N_t-1$
For $n=0$ we have the initial conditions $u=I(x)$ and $u_t=0$
At the boundaries $i=0,N_x$ we have the boundary condition $u=0$.

1.5 Step 3: Replacing derivatives by finite differences

Use a forward difference in time and a centered difference in space (*Forward Euler* scheme):

$$[D_t^+ u = \alpha D_xD_x u]_i^n \tag{9}$$

Written out,

$$\frac{u_i^{n+1}-u_i^n}{\Delta t} = \alpha\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \tag{10}$$

Initial condition: $u_i^0 = I(x_i)$, $i=0,1,\ldots,N_x$.

1.6 Step 4: Formulating a recursive algorithm

- Nature of the algorithm: compute u in space at $t=\Delta t, 2\Delta t, 3\Delta t, ...$
- Two time levels are involved in the general discrete equation: $n+1$ and n

- u_i^n is already computed for $i=0,\ldots,N_x$, and u_i^{n+1} is the unknown quantity

Solve the discretized PDE for the unknown u_i^{n+1}:

$$u_i^{n+1} = u_i^n + Fo\left(u_{i+1}^n - 2u_i^n + u_{i-1}^n\right) \tag{11}$$

where

$$Fo = \alpha\frac{\Delta t}{\Delta x^2}$$

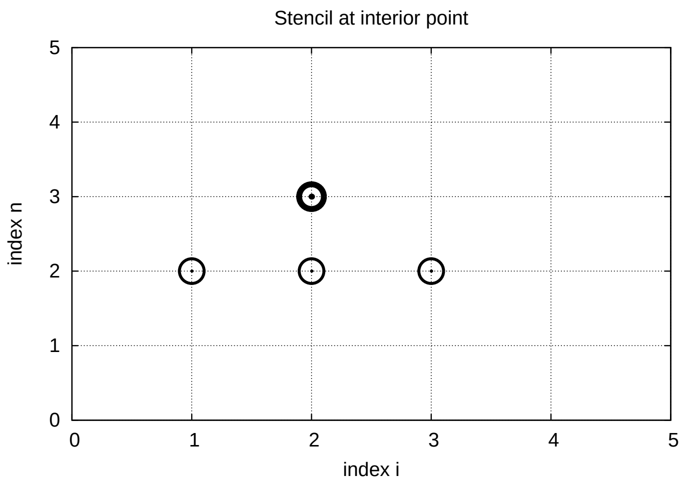
1.7 The mesh Fourier number

$$Fo = \alpha\frac{\Delta t}{\Delta x^2}$$

Observe.

There is only one parameter, Fo, in the discrete model: Fo lumps mesh parameters Δt and Δx with the only physical parameter, the diffusion coefficient α. The value Fo and the smoothness of $I(x)$ govern the quality of the numerical solution.

1.8 The finite difference stencil



1.9 The computational algorithm for the Forward Euler scheme

1. compute $u_i^0 = I(x_i)$, $i=0,\ldots,N_x$

2. for $n=0,1,\ldots,N_t$:
   (a) compute u_i^{n+1} from (11) for all the internal spatial points $i=1,\ldots,N_x-1$
   (b) set the boundary values $u_i^{n+1}=0$ for $i=0$ and $i=N_x$

Notice.

We visit one mesh point (x_i, t_{n+1}) at a time, and we have an explicit formula for computing the associated u_i^{n+1} value. The spatial points can be updated in any sequence, but the time levels t_n must be updated in cronological order: t_n before t_{n+1}.

1.10 The Python implementation of the computational algorithm

```
x = linspace(0, L, Nx+1)    # mesh points in space
dx = x[1] - x[0]
t = linspace(0, T, Nt+1)    # mesh points in time
dt = t[1] - t[0]
Fo = a*dt/dx**2
u   = zeros(Nx+1)
u_1 = zeros(Nx+1)

# Set initial condition u(x,0) = I(x)
for i in range(0, Nx+1):
    u_1[i] = I(x[i])

for n in range(0, Nt):
    # Compute u at inner mesh points
    for i in range(1, Nx):
        u[i] = u_1[i] + Fo*(u_1[i-1] - 2*u_1[i] + u_1[i+1])

    # Insert boundary conditions
    u[0] = 0;  u[Nx] = 0

    # Update u_1 before next step
    u_1[:]= u
    # or more efficient switch of references
    #u_1, u = u, u_1
```

1.11 Moving finite difference stencil

web page[1] or a movie file[2].

1.12 Demo program

- Program: `diffu1D_u0.py`[3]
- Produces animation on the screen
- Each frame stored in `tmp_frame%04d.png` files `tmp_frame0000.png`, `tmp_frame0001.png`, ...

[1] http://tinyurl.com/k3sdbuv/pub/mov-diffu/diffu1D_PDE_FE_Dirichlet_stencil_gpl/index.html
[2] http://tinyurl.com/k3sdbuv/pub/mov-diffu/diffu1D_PDE_FE_Dirichlet_stencil_gpl/movie.ogg
[3] http://tinyurl.com/jvzzcfn/diffu/diff1D_u0.py

How to make movie file in modern formats:

```
Terminal> name=tmp_frame%04d.png
Terminal> fps=8  # frames per second in movie
Terminal> avconv -r $fps -i $name -vcodec flv       movie.flv
Terminal> avconv -r $fps -i $name -vcodec libx64    movie.mp4
Terminal> avconv -r $fps -i $name -vcodec libvpx    movie.webm
Terminal> avconv -r $fps -i $name -vcodec libtheora movie.ogg
```

1.13 Forward Euler applied to an initial plug profile

$N_x = 50$. The method results in a growing, unstable solution if $Fo > 0.5$.
Choosing $Fo = 0.5$ gives a strange saw tooth-like curve.
Link to movie file[4]
Lowering Fo to 0.25 gives a smooth (expected) solution.
Link to movie file[5]

1.14 Forward Euler applied to a Gaussian profile

$N_x = 50$. $Fo = 0.5$.
Link to movie file[6]
Link to movie file[7]

1.15 Backward Euler scheme

Backward difference in time, centered difference in space:

$$[D_t^- u = D_x D_x u]_i^n \tag{12}$$

Written out:

$$\frac{u_i^n - u_i^{n-1}}{\Delta t} = \alpha \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} \tag{13}$$

Assumption: u_i^{n-1} is computed, but all quantities at the new time level t_n are unknown.

Notice.

We cannot solve wrt u_i^n because that unknown value is coupled to two other unknown values: u_{i-1}^n and u_{i+1}^n. That is, all the new unknown values are coupled to each other in a *linear system of algebraic equations*.

[4] http://tinyurl.com/k3sdbuv/pub/mov-diffu/diffu1D_u0_FE_plug/movie.ogg
[5] http://tinyurl.com/k3sdbuv/pub/mov-diffu/diffu1D_u0_FE_plug_Fo025/movie.ogg
[6] http://tinyurl.com/k3sdbuv/pub/mov-diffu/diffu1D_u0_FE_gaussian1/movie.ogg
[7] http://tinyurl.com/k3sdbuv/pub/mov-diffu/diffu1D_u0_FE_plug_gaussian2/movie.ogg

1.16 Let's write out the equations for $N_x = 3$

Equation (13) written for $i = 1, \ldots, Nx - 1 = 1, 2$ becomes

$$\frac{u_1^n - u_1^{n-1}}{\Delta t} = \alpha \frac{u_2^n - 2u_1^n + u_0^n}{\Delta x^2} \tag{14}$$

$$\frac{u_2^n - u_2^{n-1}}{\Delta t} = \alpha \frac{u_3^n - 2u_2^n + u_1^n}{\Delta x^2} \tag{15}$$

(The boundary values u_0^n and u_3^n are known as zero.)
Collecting the unknown new values on the left-hand side and writing as 2×2 matrix system:

$$\left(\begin{array}{cc} 1+2Fo & -Fo \\ -Fo & 1+2Fo \end{array}\right)\left(\begin{array}{c} u_1^n \\ u_2^n \end{array}\right) = \left(\begin{array}{c} u_1^{n-1} \\ u_2^{n-1} \end{array}\right)$$

1.17 Two classes of discretization methods: explicit and implicit

Implicit.

Discretization methods that lead linear systems are known as *implicit methods*.

Explicit.

Discretization methods that avoid linear systems and have an explicit formula for each new value of the unknown are called *explicit methods*.

1.18 The linear system for a general N_x

$$-F_o u_{i-1}^n + (1 + 2F_o)\, u_i^n - F_o u_{i+1}^n = u_{i-1}^{n-1} \tag{16}$$

for $i = 1, \ldots, Nx - 1$.
What are the unknowns in the linear system?

1. either u_i^n for $i = 1, \ldots, N_x - 1$ (all *internal* spatial mesh points)
2. or u_i^n, $i = 0, \ldots, N_x$ (all spatial points)

The linear system in matrix notation:

$$AU = b, \quad U = (u_0^n, \ldots, u_{N_x}^n)$$

1.19 A is very sparse: a tridiagonal matrix

$$
A = \begin{pmatrix}
A_{0,0} & A_{0,1} & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\
A_{1,0} & A_{1,1} & 0 & \ddots & & & & & \vdots \\
0 & A_{2,1} & A_{2,2} & A_{2,3} & \ddots & & & & \vdots \\
\vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\
\vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\
\vdots & & & 0 & A_{i,i-1} & A_{i,i} & A_{i,i+1} & \ddots & \vdots \\
\vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\
\vdots & & & & & \ddots & \ddots & \ddots & A_{N_x-1,N_x} \\
0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & A_{N_x,N_x-1} & A_{N_x,N_x}
\end{pmatrix} \tag{17}
$$

1.20 Detailed expressions for the matrix entries

The nonzero elements are given by

$$A_{i,i-1} = -F_o \tag{18}$$
$$A_{i,i} = 1 + 2F_o \tag{19}$$
$$A_{i,i+1} = -F_o \tag{20}$$

for $i = 1, \ldots, N_x - 1$.
The equations for the boundary points correspond to

$$A_{0,0} = 1, \quad A_{0,1} = 0, \quad A_{N_x,N_x-1} = 0, \quad A_{N_x,N_x} = 1$$

1.21 The right-hand side

$$
b = \begin{pmatrix} b_0 \\ b_1 \\ \vdots \\ b_i \\ \vdots \\ b_{N_x} \end{pmatrix} \tag{21}
$$

with

$$b_0 = 0 \tag{22}$$
$$b_i = u_i^{n-1}, \quad i = 1, \ldots, N_x - 1 \tag{23}$$
$$b_{N_x} = 0 \tag{24}$$

1.22 Naive Python implementation with a dense $(N_x + 1) \times (N_x + 1)$ matrix


```
x = linspace(0, L, Nx+1)   # mesh points in space
dx = x[1] - x[0]
t = linspace(0, T, N+1)    # mesh points in time
u   = zeros(Nx+1)
u_1 = zeros(Nx+1)

# Data structures for the linear system
A = zeros((Nx+1, Nx+1))
b = zeros(Nx+1)

for i in range(1, Nx):
    A[i,i-1] = -Fo
    A[i,i+1] = -Fo
    A[i,i] = 1 + 2*Fo
A[0,0] = A[Nx,Nx] = 1

# Set initial condition u(x,0) = I(x)
for i in range(0, Nx+1):
    u_1[i] = I(x[i])

import scipy.linalg

for n in range(0, Nt):
    # Compute b and solve linear system
    for i in range(1, Nx):
        b[i] = -u_1[i]
    b[0] = b[Nx] = 0
    u[:] = scipy.linalg.solve(A, b)

    # Update u_1 before next step
    u_1, u = u, u_1
```

1.23 A sparse matrix representation will dramatically reduce the computational complexity

- With a dense matrix, the algorithm leads to $\mathcal{O}(N_x^3)$ operations
- Utilizing the sparsity, the algorithm has complexity $\mathcal{O}(N_x)$!
- `scipy.sparse` enables storage and calculations with the three nonzero diagonals only

```
# Representation of sparse matrix and right-hand side
diagonal  = zeros(Nx+1)
lower     = zeros(Nx)
upper     = zeros(Nx)
b         = zeros(Nx+1)
```

1.24 Computing the sparse matrix

```
# Precompute sparse matrix
diagonal[:] = 1 + 2*Fo
lower[:] = -Fo  #1
upper[:] = -Fo  #1
# Insert boundary conditions
diagonal[0] = 1
upper[0] = 0
```

```
diagonal[Nx] = 1
lower[-1] = 0

import scipy.sparse
A = scipy.sparse.diags(
    diagonals=[main, lower, upper],
    offsets=[0, -1, 1], shape=(Nx+1, Nx+1),
    format='csr')

# Set initial condition
for i in range(0,Nx+1):
    u_1[i] = I(x[i])

for n in range(0, Nt):
    b = u_1
    b[0] = b[-1] = 0.0  # boundary conditions
    u[:] = scipy.sparse.linalg.spsolve(A, b)
    # Switch variables before next step
    u_1, u = u, u_1
```

1.25 Backward Euler applied to a plug profile

$N_x = 50$. $Fo = 0.5$.
Link to movie file[8]

1.26 Backward Euler applied to a Gaussian profile

$N_x = 50$.
Link to movie file[9]
$Fo = 5$.
Link to movie file[10]

1.27 Crank-Nicolson scheme

The PDE is sampled at points $(x_i, t_{n+\frac{1}{2}})$ (at the spatial mesh points, but in between two temporal mesh points).

$$\frac{\partial}{\partial t}u(x_i, t_{n+\frac{1}{2}}) = \alpha\frac{\partial^2}{\partial x^2}u(x_i, t_{n+\frac{1}{2}})$$

for $i = 1, \ldots, N_x - 1$ and $n = 0, \ldots, N_t - 1$.
Centered differences in space and time:

$$[D_t u = \alpha D_x D_x u]_i^{n+\frac{1}{2}}$$

1.28 Averaging in time is necessary in the Crank-Nicolson scheme

Right-hand side term:

$$\frac{1}{\Delta x^2}\left(u_{i-1}^{n+\frac{1}{2}} - 2u_i^{n+\frac{1}{2}} + u_{i+1}^{n+\frac{1}{2}}\right)$$

[8]http://tinyurl.com/k3sdbuv/pub/mov-diffu/diffu1D_u0_BE_plug/movie.ogg
[9]http://tinyurl.com/k3sdbuv/pub/mov-diffu/diffu1D_u0_BE_gaussian1/movie.ogg
[10]http://tinyurl.com/k3sdbuv/pub/mov-diffu/diffu1D_u0_BE_gaussian1_Fo5/movie.ogg

Problem: $u_i^{n+\frac{1}{2}}$ is not one of the unknowns we compute.
Solution: replace $u_i^{n+\frac{1}{2}}$ by an arithmetic average:

$$u_i^{n+\frac{1}{2}} \approx \frac{1}{2}\left(u_i^n + u_i^{n+1}\right)$$

In compact notation (arithmetic average in time $\overline{u}^t$):

$$[D_t u = \alpha D_x D_x \overline{u}^t]_i^{n+\frac{1}{2}}$$

1.29 Crank-Nicolsoon scheme written out

$$u_i^{n+1} - \frac{1}{2}Fo(u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}) = u_i^n + \frac{1}{2}Fo(u_{i-1}^n - 2u_i^n + u_{i+1}^n) \tag{25}$$

Observe:

- The unknowns are $u_{i-1}^{n+1}, u_i^{n+1}, u_{i+1}^{n+1}$
- These unknowns are coupled to each other (in a linear system)
- Must solve $AU = b$ at each time level

Now,

$$A_{i,i-1} = -\frac{1}{2}F_o \tag{26}$$
$$A_{i,i} = \frac{1}{2} + F_o \tag{27}$$
$$A_{i,i+1} = -\frac{1}{2}F_o \tag{28}$$

for internal points. For boundary points,

$$A_{0,0} = 1 \tag{29}$$
$$A_{0,1} = 0 \tag{30}$$
$$A_{N_x,N_x-1} = 0 \tag{31}$$
$$A_{N_x,N_x} = 1 \tag{32}$$

Right-hand side:

$$b_0 = 0 \tag{33}$$
$$b_i = u_i^{n-1}, \quad i = 1, \ldots, N_x - 1 \tag{34}$$
$$b_{N_x} = 0 \tag{35}$$

1.30 Crank-Nicolson applied to a plug profile

Crank-Nicolson never blows up, so any Fo can be used (modulo loss of accuracy).

$N_x = 50$. $Fo = 5$ gives instabilities.

Link to movie file[11]

$N_x = 50$. $Fo = 0.5$ gives a smooth solution.

Link to movie file[12]

1.31 Crank-Nicolson applied to a Gaussian profile

$N_x = 50$.

Link to movie file[13]

$Fo = 5$.

Link to movie file[14]

1.32 The θ rule

The θ rule condenses a family of finite difference approximations in time to one formula

- $\theta = 0$ gives the Forward Euler scheme in time
- $\theta = 1$ gives the Backward Euler scheme in time
- $\theta = \frac{1}{2}$ gives the Crank-Nicolson scheme in time

Applied to $u_t = \alpha u_{xx}$:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \alpha\left(\theta\frac{u_{i+1}^{n+1} - 2u_i^{n+1} + u_{i-1}^{n+1}}{\Delta x^2} + (1-\theta)\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}\right)$$

Matrix entries:

$$A_{i,i-1} = -F_o\theta, \quad A_{i,i} = 1 + 2F_o\theta \quad , A_{i,i+1} = -F_o\theta$$

Right-hand side:

$$b_i = u_i^n + F_o(1-\theta)\frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}$$

1.33 The Laplace and Poisson equation

Laplace equation:

$$\nabla^2 u = 0, \quad \text{1D: } u''(x) = 0$$

Poisson equation:

$$-\nabla^2 u = f, \quad \text{1D: } -u''(x) = f(x)$$

These are limiting behavior of time-dependent diffusion equations if

[11] http://tinyurl.com/k3sdbuv/pub/mov-diffu/diffu1D_u0_theta_plug_Fo5/movie.ogg
[12] http://tinyurl.com/k3sdbuv/pub/mov-diffu/diffu1D_u0_theta_plug/movie.ogg
[13] http://tinyurl.com/k3sdbuv/pub/mov-diffu/diffu1D_u0_theta_gaussian1/movie.ogg
[14] http://tinyurl.com/k3sdbuv/pub/mov-diffu/diffu1D_u0_theta_gaussian1_Fo5/movie.ogg

$$\lim_{t\to\infty}\frac{\partial u}{\partial t} = 0$$

Then $u_t = \alpha u_{xx} + 0$ in the limit $t \to \infty$ reduces to

$$u_{xx} + f = 0$$

1.34 We can solve 1D Poisson/Laplace equation by going to infinity in time-dependent diffusion equations

Looking at the numerical schemes, $Fo \to \infty$ leads to the Laplace or Poisson equations (without f or with f, resp.).

Good news: choose Fo large in the BE or CN schemes and *one time step* is enough to produce the stationary solution for $t \to \infty$.

1.35 Extensions

These extensions are performed exactly as for a wave equation as they only affect the spatial derivatives (which are the same as in the wave equation).

- Variable coefficients
- Neumann and Robin conditions
- 2D and 3D

Future versions of this document will for completeness and independence of the wave equation document feature info on the three points. The Robin condition is new, but straightforward to handle:

$$-\alpha\frac{\partial u}{\partial n} = h_T(u - U_s), \quad [-\alpha D_x u = h_T(u - U_s)]_i^n$$

2 Analysis of schemes for the diffusion equation

2.1 Properties of the solution

The PDE

$$u_t = \alpha u_{xx} \tag{36}$$

admits solutions

$$u(x,t) = Qe^{-\alpha k^2 t}\sin(kx) \tag{37}$$

Observations from this solution:

- The initial shape $I(x) = Q\sin kx$ undergoes a damping $\exp(-\alpha k^2 t)$
- The damping is very strong for short waves (large k)
- The damping is weak for long waves (small k)
- Consequence: u is smoothened with time

2.2 Example

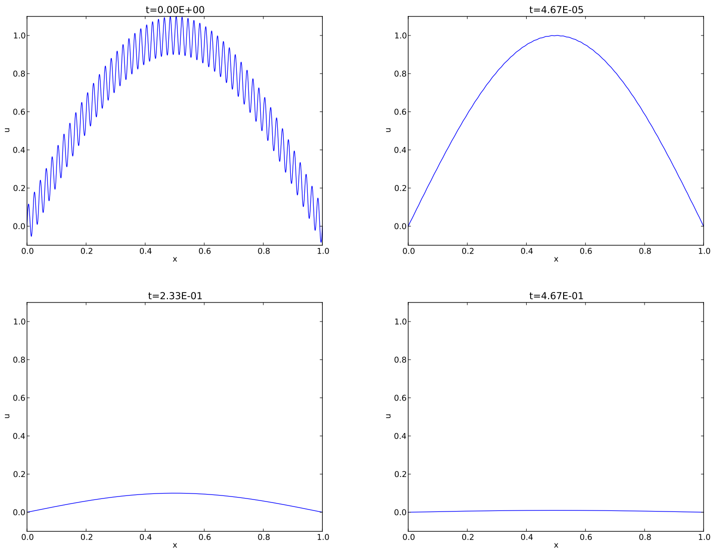
Test problem:

$$
\begin{aligned}
u_t &= u_{xx}, & x \in (0,1),\ t \in (0,T] \\
u(0,t) &= u(1,t) = 0, & t \in (0,T] \\
u(x,0) &= \sin(\pi x) + 0.1\sin(100\pi x)
\end{aligned}
$$

Exact solution:

$$u(x,t) = e^{-\pi^2 t}\sin(\pi x) + 0.1e^{-\pi^2 10^4 t}\sin(100\pi x) \tag{38}$$

2.3 Visualization of the damping in the diffusion equation



2.4 Damping of a discontinuity; problem and model

Problem.

Two pieces of a material, at different temperatures, are brought in contact at $t = 0$. Assume the end points of the pieces are kept at the initial temperature. How does the heat flow from the hot to the cold piece?

Solution.

Assume a 1D model is sufficient (insulated rod):

$$u(x,0) = \begin{cases} U_L, & x < L/2 \\ U_R, & x \geq L/2 \end{cases}$$

$$\frac{\partial u}{\partial t} = \alpha\frac{\partial^2 u}{\partial x^2}, \quad u(0,t) = U_L,\ u(L,t) = U_R$$

2.5 Damping of a discontinuity; Backward Euler simulation

Movie[15]

2.6 Damping of a discontinuity; Forward Euler simulation

Movie[16]

2.7 Damping of a discontinuity; Crank-Nicolson simulation

Movie[17]

2.8 Fourier representation

Represent $I(x)$ as a Fourier series

$$I(x) \approx \sum_{k\in K} b_k e^{ikx} \tag{39}$$

The corresponding sum for u is

$$u(x,t) \approx \sum_{k\in K} b_k e^{-\alpha k^2 t} e^{ikx}\,. \tag{40}$$

Such solutions are also accepted by the numerical schemes, but with an amplification factor A different from $\exp{(-\alpha k^2 t)}$:

$$u_q^n = A^n e^{ikq\Delta x} = A^n e^{ikx} \tag{41}$$

2.9 Analysis of the finite difference schemes

Stability:

- $|A| < 1$: decaying numerical solutions (as we want)
- $A < 0$: *oscillating* numerical solutions (as we do not want)

Accuracy:

- Compare numerical and exact amplification factor: A vs $A_{\mathrm{e}} = \exp{(-\alpha k^2\Delta t)}$

[15] http://tinyurl.com/k3sdbuv/pub/mov-diffu/BE_C0.5/index.html
[16] http://tinyurl.com/k3sdbuv/pub/mov-diffu/FE_C0.5/index.html
[17] http://tinyurl.com/k3sdbuv/pub/mov-diffu/CN_C5/index.html

2.10 Analysis of the Forward Euler scheme

$$[D_t^+ u = \alpha D_x D_x u]_q^n$$

Inserting

$$u_q^n = A^n e^{ikq\Delta x}$$

leads to

$$A = 1 - 4C\sin^2\left(\frac{k\Delta x}{2}\right), \quad C = \frac{\alpha\Delta t}{\Delta x^2} \tag{42}$$

The complete numerical solution is

$$u_q^n = (1 - 4C\sin^2 p)^n e^{ikq\Delta x}, \quad p = k\Delta x/2 \tag{43}$$

2.11 Results for stability

We always have $A \leq 1$. The condition $A \geq -1$ implies

$$4C\sin^2 p \leq 2$$

The worst case is when $\sin^2 p = 1$, so a sufficient criterion for stability is

$$C \leq \frac{1}{2} \tag{44}$$

or:

$$\Delta t \leq \frac{\Delta x^2}{2\alpha} \tag{45}$$

Implications of the stability result.

Less favorable criterion than for $u_{tt} = c^2 u_{xx}$: halving Δx implies time step $\frac{1}{4}\Delta t$ (not just $\frac{1}{2}\Delta t$ as in a wave equation). Need very small time steps for fine spatial meshes!

2.12 Analysis of the Backward Euler scheme

$$[D_t^- u = \alpha D_x D_x u]_q^n$$

$$u_q^n = A^n e^{ikq\Delta x}$$

$$A = (1 + 4C\sin^2 p)^{-1} \tag{46}$$

$$u_q^n = (1 + 4C\sin^2 p)^{-n} e^{ikq\Delta x} \tag{47}$$

2.13 Stability

We see from (46) that $|A| < 1$ for all $\Delta t > 0$ and that $A > 0$ (no oscillations).

2.14 Analysis of the Crank-Nicolson scheme

The scheme

$$[D_t u = \alpha D_x D_x \overline{u}^x]_q^{n+\frac{1}{2}}$$

leads to

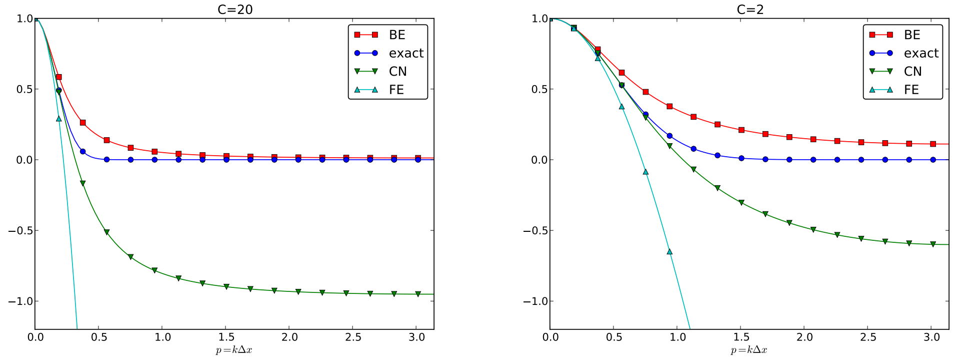
$$A = \frac{1 - 2C\sin^2 p}{1 + 2C\sin^2 p} \tag{48}$$

$$u_q^n = \left(\frac{1 - 2C\sin^2 p}{1 + 2C\sin^2 p}\right)^n e^{ikp\Delta x} \tag{49}$$

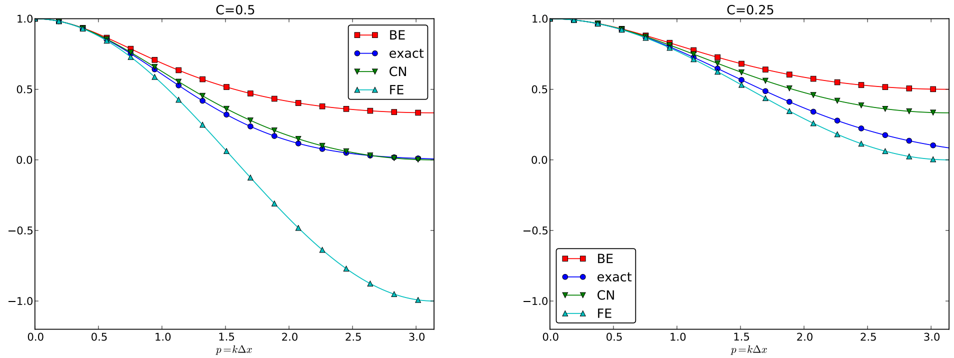
2.15 Stability

The criteria $A > -1$ and $A < 1$ are fulfilled for any $\Delta t > 0$.

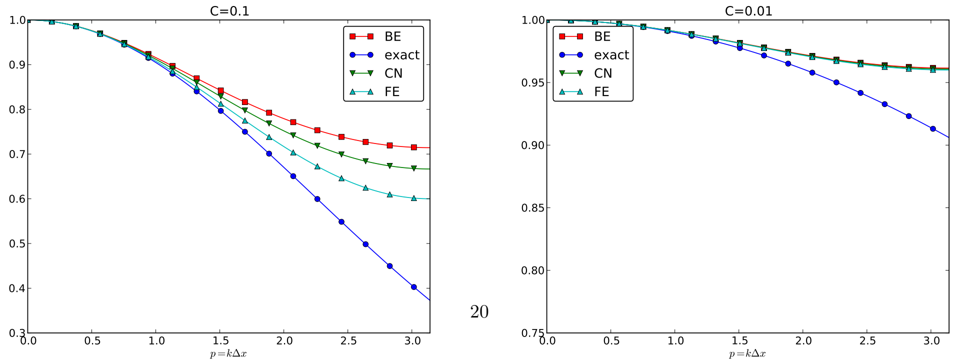
2.16 Summary of accuracy of amplification factors; large time steps



2.17 Summary of accuracy of amplification factors; time steps around the Forward Euler stability limit



2.18 Summary of accuracy of amplification factors; small time steps



2.19 Observations

- Crank-Nicolson gives oscillations and not much damping of short waves for increasing C.

- These waves will manifest themselves as high frequency oscillatory noise in the solution.
- All schemes fail to dampen short waves enough

The problems of correct damping for $u_t = u_{xx}$ is partially manifested in the similar time discretization schemes for $u'(t) = -\alpha u(t)$.

Index